1994

A Heuristic Approach for Shortest Path Problem With Rectilinear Obstacles.

Joon Shik Lim
Louisiana State University and Agricultural & Mechanical College

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9502127

A heuristic approach for shortest path problem with rectilinear obstacles

Lim, Joon Shik, Ph.D.

The Louisiana State University and Agricultural and Mechanical Col., 1994

U·M·I

300 N. Zeeb Rd.
Ann Arbor, MI 48106

A HEURISTIC APPROACH FOR SHORTEST PATH PROBLEM WITH RECTILINEAR OBSTACLES

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in Partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by

Joon Shik Lim
B.S., Inha University, Korea, 1986
M.S., University of Alabama at Birmingham, 1989
May 1994

ACKNOWLEDGMENTS

I am indebted to my advisor, Dr. S. S. Iyengar, and Dr. Si-Qing Zheng for their invaluable support, positive encouragement, and guidance. I would like to thank to other committee members, Dr. K. Tang, Dr. D. Kraft, Dr. B. Jones, Dr. J. Chen, Dr. J. Madden.

I wish to thank my parents and my wife for their immeasurable love, encouragement, and support of my educational endeavors.

In addition, many enjoyable and valuable discussions with Mr. Y. Cho and Mr. M. Yoo in the Department of Computer Science is appreciated.

Finally, I would like to thank to God who has kept my lovely family, Myung A., Sung Bin, and me, throughout my graduate study period in LSU.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

This dissertation presents new heuristic search algorithms, the *Guided Minimum Detour* (*GMD*) algorithm and the *Line-by-Line Guided Minimum Detour* (*LGMD*) algorithm for searching rectilinear (L_1) shortest paths in the presence of rectilinear obstacles. The *GMD* algorithm combines the best features of maze-running algorithms and line-search algorithms. The *LGMD* algorithm is a modification of the *GMD* algorithm that improves on efficiency using line-by-line extensions. Our *GMD* and *LGMD* algorithms always find a rectilinear shortest path using the *guided* A^* search method without constructing a *connection graph* that contains a shortest path. The *GMD* algorithm and the *GMD* algorithm can be implemented in $O(m+(e+N)\log e)$ and $O((e+N)\log e)$ time, respectively, and $O(e+N)$ space, where m is the total number of searched nodes, e is the number of boundary sides of obstacles, and N is the total number of searched line segments. We consider the problem of finding a shortest path in terms of the number of bends and the combined length and bends.

CHAPTER 1

INTRODUCTION

The problem of finding a shortest path in the presence of rectilinear obstacles has applications in *robotics*, *VLSI design*, and *geographical information systems* [13]. In VLSI design, there are two basic classes of sequential algorithms: *maze-running* algorithms and *line-search* algorithms. These algorithms are aimed mostly at finding an obstacle-avoiding path, preferably the shortest one, between two given points. The maze-running algorithms can be characterized as target-directed grid extension. The first such algorithm is *Lee* algorithm [12], which is an application of the *breadth-first* shortest path search algorithm. The major disadvantage of the original Lee algorithm is that it requires $O(n^2)$ memory and running time in the worst case for nXn grid graphs. In addition, each node requires $O(\log L)$ bits, where L is the length of the shortest path from a source node s to a target node t. It is desirable to reduce the memory requirement for each node. More important, the size of search space must be reduced, since the running time is proportional to this size. There are a large number of variations (e.g., [1][6][7][8][10] [13][14][17][18][19][20][22][23]) of the

original Lee algorithm. Akers [1] modified the Lee algorithm by introducing a coding scheme, which requires two bits per node regardless of the value L. Hart et al. [8] proposed the idea of using a lower bound on the Manhattan distance between a source node and a target node. Hadlock applied this to the shortest path algorithm, called *Minimum Detour algorithm* [7]. For each node, he used a new labeling method called *detour number* which is the total number of nodes moves away from a target node t. For a path from the source node s to the target node t with detour number d, its length is $M(s,t)+2d$, $d \geq 0$, where $M(s,t)$ is the Manhattan distance between s and t. The minimum detour algorithm searches a shortest path by minimizing the detour number d. Since $M(s,t)$ is fixed for a given pair (s,t), a path from s to t is a shortest one if d is minimum. The minimum detour algorithm guarantees finding the shortest path using time between $O(n)$ and $O(n^2)$ for nXn grid graphs. Although a depth-first search method usually finds suboptimal paths in the grid graph, the method is useful to reduce search space compared with a breadth-first search method. Soukup [23] incorporated the depth-first search with the breadth-first search to reduce search space and time. This algorithm guarantees finding a path if it exists, but not necessarily the shortest one. The Soukup algorithm executes the depth-first search from

the *s* toward *t* using a "*don't change direction*" heuristic until an obstacle is hit or the target node *t* is reached. If an obstacle is hit, then the breadth-first search is used for searching around the obstacle until a grid node directs toward the target node *t* is found, and this procedure is repeated until the target node *t* is reached.

All partial paths generated by maze-running algorithms are represented by unit grid line segments. These algorithms are still considered memory-and-time inefficient. Line-search algorithms have been proposed to achieve improved performance. Since such algorithms search a path as a sequence of line segments of variable length, they save memory and quickly find a simple-shaped path. The major drawback of the line-search algorithms is that they usually do not guarantee finding a shortest path. The idea behind these algorithms is to reduce the size of representation for all searched grid nodes by a set of long line segments. The firsts of such algorithms are reported in [9] and [15]. The line-search algorithm given in [9] is similar to the one in [15]. The difference is that the algorithm in [9] generates significantly fewer trial lines at every level. Several recent line-search algorithms (e.g., [4][13][16][21][24]) are based on powerful computational geometry techniques. Wu et al. [24] introduced a rather small connection graph, the *track*

graph, which contains the shortest path, but it is not a strong connection graph, i.e., the track graph may not contain a shortest path between a pair of two points. The run time of their algorithm is $O((e+k)\log t)$, where e is the total number of boundary sides of obstacles, t is the total number of extreme edges of all obstacles, and k is the number of intersections among obstacle tracks, which is bounded by $O(t^2)$. Zheng et al. [26] proposed an efficient geometric algorithm for constructing a connection graph G_c. They presented a framework for designing a class of time-and-space efficient rectilinear shortest path and rectilinear minimum spanning tree algorithms based on G_c. De Rezende et al. [21] considered a special case that all obstacles are rectangles. Their algorithm constructs a strong connection graph and finds a shortest path from s to t in time $O(n\log n)$, where n is the number of obstacles. Clarkson et al. [4] generalized the shortest path problem to the case of arbitrarily shaped obstacles. Their algorithm runs in time $O(n\log^2 n)$. For the special case where obstacles are just rectilinear line segments, Berg et al. [2] studied the shortest path problem in a combined metric that generalizes the L_1 metric and the rectilinear link metric. A good survey of algorithms for the rectilinear shortest path problem can be found in [13]. Most of these line-search algorithms can find a shortest

path with subquadratic time and memory in the worst cases using a connection graph that contains the shortest paths. Heuristic algorithms, however, may still perform in practice better for the shortest path problems.

In this dissertation, we introduce two new heuristic algorithms, the *Guided Minimum Detour* (*GMD*) algorithm and the *Line-by-Line Guided Minimum Detour* (*LGMD*) algorithm. The *GMD* algorithm incorporates the best features of maze-running algorithms and line-search algorithms. The algorithm constructs node by node a path from a source node to a target node. This maze-running feature guarantees that the *GMD* algorithm always finds the shortest path if one exists. The *GMD* algorithm uses a heuristic search method called *guided* A^*. It is the A^* search [8] with the heuristic "*don't change direction.*" In addition, the underlying data structure used for storing the intermediate results maintains extended line segments on a grid graph that is much sparser than the original grid search graph. These line segments are characterized by a set of special grid points called *base nodes*. The technique described in this paper reduces space, compared with the existing maze-running algorithms. On the basis of *GMD* algorithm, we present a modified algorithm called the *LGMD* algorithm. Both the *GMD* algorithm and the *LGMD* algorithm do not need a connection graph. The *LGMD* algorithm is a line-search

algorithm, which improves on the existing *GMD* algorithm's drawback--the running time--without losing solution optimality. In the worst case, our *GMD* and *LGMD* algorithms have the time and space complexities comparable to those of existing algorithms. In most cases, however, our algorithms provide significant time and space improvements.

1.1. Maze-Running Algorithms

Since our algorithm is based on the existing maze-running algorithms, it is instructive to briefly describe some of these algorithms. Such a survey, which is not intended to be complete, is important in comparing our algorithm with previously known results.

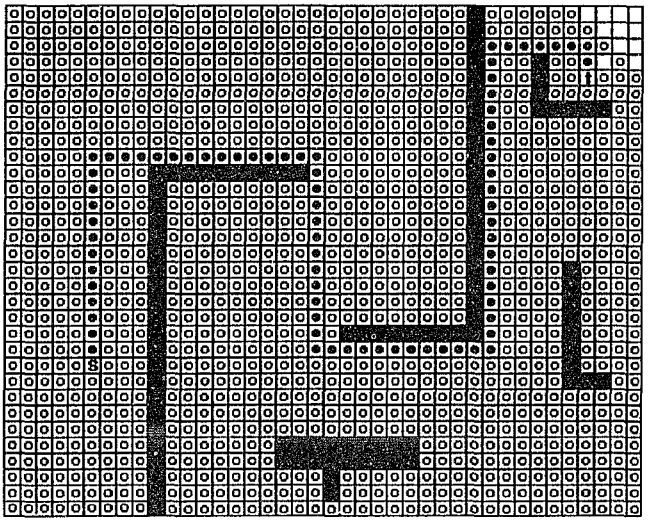
The Lee algorithm [12] is usually referred to as a wave propagation method. This algorithm consists of these phases: *search*, *path trace*, and *label clearance*. In the search phase, the cells are labeled in a systematic way. Initially, a label "1" is entered in every available cell adjacent to the cell containing the source node s. Then, a label "2" is entered in every available cell adjacent to these labeled "1"; and so on. Such a process is continued until either the cell containing the target node t is reached, or in the kth iteration no available cell adjacent to those labeled "k-1" exists. In the former case, a path from node s to node t is found. The latter case indicates that no path from s to t exists. This search can be viewed

as a *breadth-first-search*, and it is similar to the movement of the wave front created by dropping a pebble into a pool of water.

When the target node is reached in the search phase, path trace phase is followed. By tracing the labeled cells in descending order from t to s, a shortest path is obtained. Finally, all labeled cells except these that make the founded path are unlabeled. This process is called *label clearance*, which is almost the same as the search phase.

The Lee algorithm requires $O(n^2)$ memory for an $n \times n$ grid. In addition, each node requires $O(\log L)$ bits where L is the length of a shortest path from s to t. It requires $O(L^2)$ running time, and $O(n^2)$ in the worst case. It is desirable to reduce the memory requirement for each node. More importantly, the size of search space must be reduced, since the running time is proportional to this size.

Akers [1] modified the Lee algorithm by introducing a coding scheme, which requires two bits per node regardless of the value L. Examples of previous efforts in reducing size of search space are discussed in the following subsections. Figure 1 shows an example of extended nodes for Lee algorithm.



s: Source Node t: Target Node o: Extended Nodes

Figure 1. Expanded Nodes for Lee Algorithm

Hart et. al [8] proposed the idea using a lower bound on the Manhattan distance between an source node and a target node. Hadlock applied this to the shortest path algorithm, called *Minimum Detour* algorithm. He used a new labeling method, called *detour number*, for each node. To present the length of path from its source node s to its target node t, the detour number d that is the total number of times the path moves away from its target node t, and the Manhattan distance M are used such that:

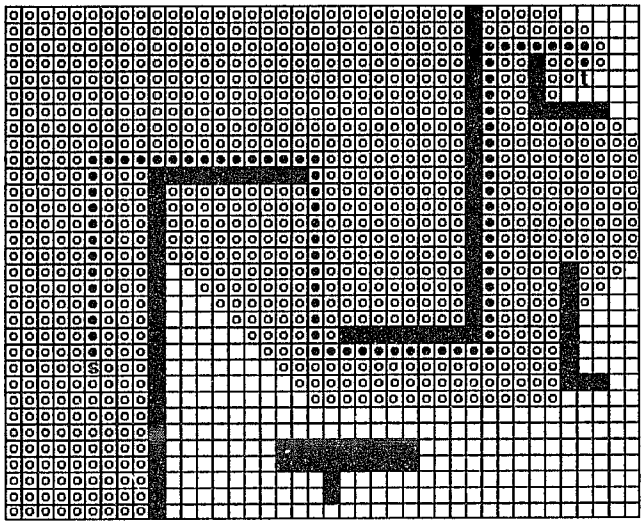
$$M(s, t) + 2d, \text{ where } d \geq 0.$$

The minimum detour algorithm searches its shortest path using the minimized detour number d. A path length from s to t is the shortest one when d is the minimum

detour number of the path with respect to t, since the path length $M(s,t)+2d$ is constant and d is minimized. In this algorithm, the wave-front nodes are classified to two classes; one contains the positive nodes that move toward a target node t to be put on *P-stack*, the other one contains the negative nodes that move away from the node t to be put on *N-stack*. The positive wave-front nodes in *P-stack* are unstacked and expanded to their neighbors. Then the expanded nodes are put on *P-stack* or *N-stack*. If *P-stack* is empty, all nodes in *N-stack* are moved to *P-stack* and the expansion is repeated until a target node t is found. Figure 2 shows extended nodes for Hadlock algorithm.

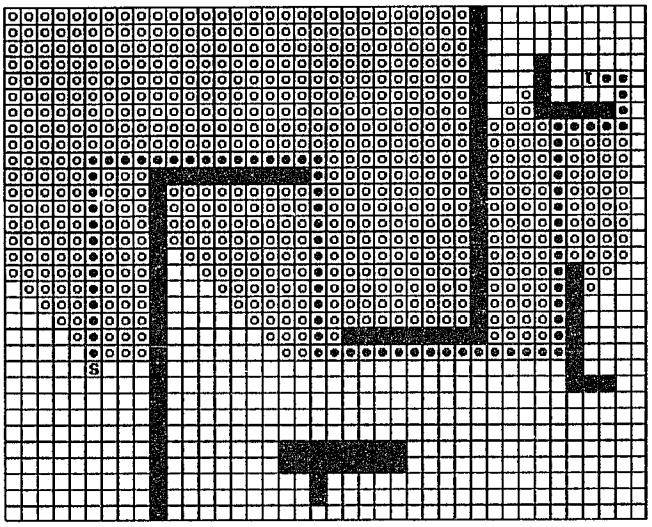
The minimum detour algorithm guarantees to find the shortest path using time between $O(n)$ and $O(n^2)$ for an $n \times n$ grid plane. Figure 2 shows how the same problem in previous section is solved using Hadlock's algorithm.

Although *depth-first-search* methods usually find suboptimal paths on a grid graph, the methods are useful to reduce search space compared with *breadth-first-search* methods. Soukup [23] incorporates a *depth-first-search* with a *breadth-first-search* in order to reduce search space and time. This algorithm guarantees to find a path if there exists, but not necessarily an optimal path. Initially, the Soukup algorithm executes a *depth-first-search* from a source node toward a target node using "*don't*



s: Source Node t: Target Node o: Extended Nodes

Figure 2. Expanded Nodes for Hadlock's *MD* Algorithm



s: Source Node t: Target Node o: Extended Nodes

Figure 3. Expanded Nodes for Soukup's Fast Maze Router Algorithm

change direction" heuristic until an obstacle is hit or a target node is found. If an obstacle is hit, then a *breadth-first-search* is used for searching around the obstacle until a node directs to a target node, and above procedures are repeated. Figure 3 shows how the same problem in the previous example is solved using Soukup's Fast Maze Router algorithm.

1.2. Line-search Algorithms

All above mentioned algorithms and many of their variations are based on grid expansion. They are called *maze-running* algorithm. Since data must be kept for each grid nodes, the memory requirements this class of algorithms are excessive. Another class of path-finding algorithms, referred to as *line-search* algorithms aim at reducing memory resources required. The idea behind these algorithms is to eliminate the representation of all available grid nodes by representing the search space and paths with a set of line segments. The first of such algorithms is reported in [9] and [15]. The basic operations of algorithm of [15] are as follows. First, straight lines are emanated from a source node s to a target node t in four directions. These search lines are called *level-0 trial lines* and stored in a temporary storage. Then, the path search is conducted by a iterating process. At the ith iteration step, the following

operations are performed: Pick up *level-i trial lines* one by one from the temporary storage. Along each such trial line, trace all grid nodes. Emanate new lines perpendicular to the trial line from these base nodes. These newly generated line segments, which end either at the boundary of an obstacle or the boundary of the grid, are identified as *level-*(i+1) *trial lines*. All *level-*(i+1) *trial lines* are stored in a temporary storage. This process continues until a trial line from s meets a trial line from t. This algorithm finds a path from s to t if there exists one, but the path is not generated to be the shortest one. The line-search algorithm given in [9] is similar to the one in [15]. The difference is that the algorithm in [9] generates significantly less trial lines at every level. It requires less memory, but does not necessarily find a path from s to t even such a path exists. Several recent line-search algorithms are based on powerful computational geometry techniques. These algorithms can find shortest path using subquadratic time and memory in the worst cases. However, heuristic algorithms may still perform better for most real routing problems in practice. The examples for the line_search algorithms in [9] and [15] are shown in Figure 4 and Figure 5, respectively.

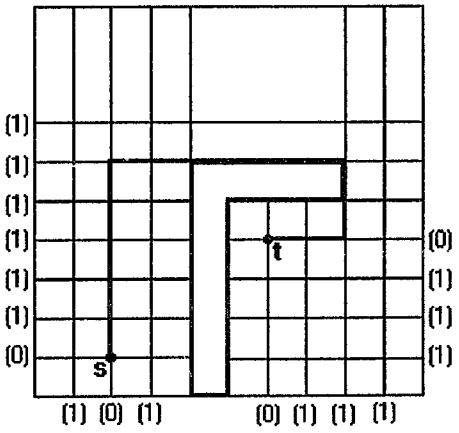


Figure 4. Line Search Algorithm of Mikami and Tabuchi

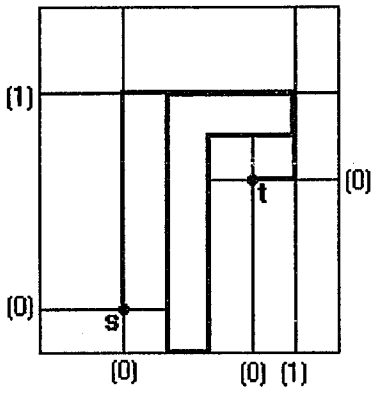


Figure 5. Modified Version of the Algorithm by Mikami and Tabuchi

CHAPTER 2

A NEW ALGORITHM: GUIDED MINIMUM DETOUR ALGORITHM (*GMD*)

Let G be an $n \times n$ uniform grid graph that consists of a set of *grid nodes* $\{(x,y) \mid x$ and y are integers such that $1 \le x,y \le n\}$ and *grid edges* connecting grid nodes that are unit distance apart. The length between any two adjacent grid nodes in G is assumed to be 1. A horizontal (vertical) grid line segment is a path consists of horizontal (vertical) grid edges. Let $B=\{B_1, B_2, \ldots, B_p,\}$ be a set of mutually disjoint rectilinear simple polygons with boundaries on G. Each polygon in B is an obstacle. Let G' denote a partial grid of G that consists of grid nodes that are not contained in the interior of any obstacle in B, and grid edges that are not incident to interior grid nodes of any obstacle in B (see Figure 6.a).

Let H be an $(n+1)\times(n+1)$ grid node set $\{(x,y) \mid x=i-0.5$, $y=j-0.5$, i and j are integers such that $1 \le i,j \le n+1\}$ and grid edges connecting grid nodes that are unit distance apart. Each face formed by four grid nodes of H is called a *cell*. We define the *offset representation* H' of G' as the portion of grid H with all cells in the interior of portions corresponding obstacles in B removed (see Figure 6.b). G and H are equivalent. We use the offset

representation to demonstrate the maze-running features of the *GMD* algorithm in our figures.

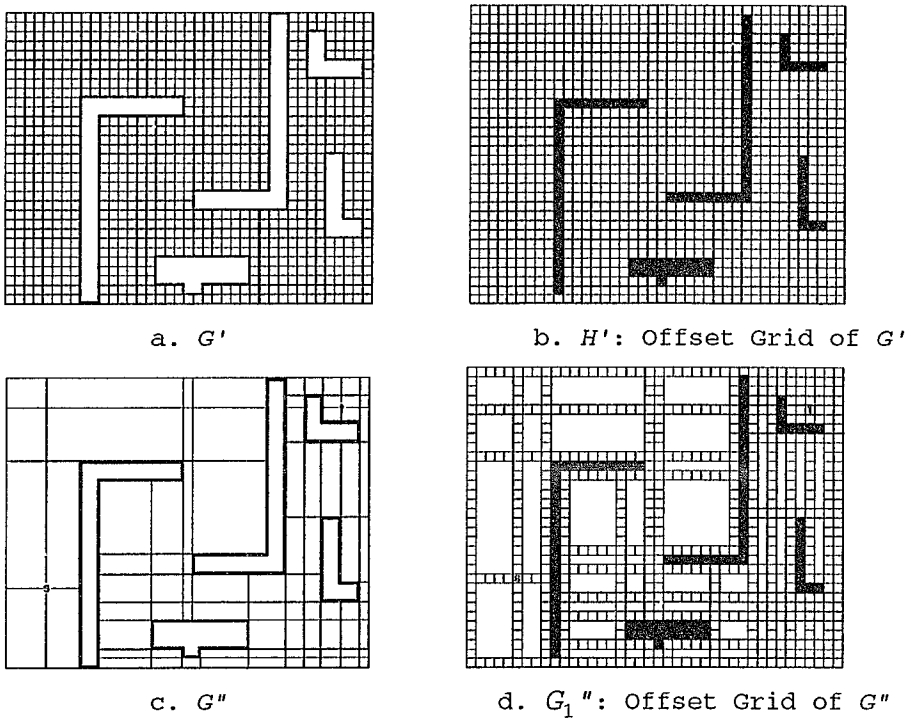
To simplify presentation and analysis, we construct a grid structure G'' from G' as follows (see Figure 6.c). Define a horizontal (vertical) line segment $l=(u,v)$ in G' as a *maximal horizontal (vertical) line segment* of G' if l does not cross any B_j in B, and u and v are the only two points on l that are on the boundaries of G or obstacles in B. Let $HL(G')=\{l \mid l=(u,v)$ is a maximal vertical line segment of G' such that at least one of its endpoints u and v is a corner of some B_i in $B\}$ and $VL(G')=\{l \mid l=(u,v)$ is a maximal vertical line segment of G' such that at least one of its endpoints u and v is a corner of some B_i in $B\}$. Let $L(G',B)$ be the set of line segments that form the boundaries of G and obstacles in B. Let L_s be the set of all maximal line segments that include s and L_t be the set of all maximal line segments on the lines passing through t. The nodes of G'' are the intersection points of the line segments in $L(G',B) \cup HL(G') \cup VL(G') \cup L_s \cup L_t$, and the edges of G'' are the subsegments generated by the intersections. Let $|L(G',B)|=e$. Clearly, G'' is a graph much sparser than G' in most cases, since the numbers of nodes and edges in G'' are at most $O(e^2)$. Consider any path P' from s to t in G'. It is easy to verify that, starting from s, one can "bend" P' to obtain a modified path P'' in G'' such that the length

of P'' is no larger than the length of P'. Therefore, we have the following lemma.

Lemma 1. If there exists a path from *s* to *t* in *G'*, then the shortest path from *s* to *t* in *G''* is a shortest path from *s* to *t* in *G'*.

By this fact, the search of a shortest path can be restricted to G''. As some existing algorithms (e.g., the ones in [21][24][25]), our *GMD* and *LGMD* algorithms search a shortest path in an implicit connection graph, which is the reduced grid graph G''. A refined feature of our algorithms is that, unlike previous algorithms that generate a reduced search graph (track graph or connection graph) prior to a search, our underlying search graph G'' is never explicitly constructed.

Let G_1'' denote the grid graph obtained from G'' by adding grid nodes to G'' such that any two adjacent nodes are unit distance apart (see Figure 6.d). Clearly, G_1'' is a subgraph of G'. With respect to G_1'', a directed path P is represented by $P(v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_m)$ with node set $\{v_1, v_2, \ldots, v_m\}$ and edge set $\{(v_i, v_{i+1}): i=1, \ldots, m-1\}$. If v_1 is a neighbor grid node of v_2, the edge $v_1 \rightarrow v_2$ is called a *unit line segment* with length 1. The length of P, denoted by $L(P)$, is $m-1$.



a. G' b. H': Offset Grid of G'

c. G'' d. G_1'': Offset Grid of G''

Figure 6. Definitions of the Grid Graphs

For any path P in G_1'', the *detour length* of P, denoted by $DL(P)$, is the total number of grid nodes that proceed away from the target node t in P. For a line segment $u \rightarrow v$ in $P=(s \rightarrow \ldots \rightarrow u \rightarrow v \rightarrow \ldots \rightarrow t)$, $DL[u \rightarrow v]$ is a detour length of the subpath $P_s=(s \rightarrow \ldots \rightarrow u \rightarrow v)$, i.e., $DL[u \rightarrow v]=DL(P_s)$. Let $M(s,t)$ denote the Manhattan Distance between s and t in G_1''. Clearly, $L(P)=M(s,t)+2 \cdot DL(P)$ is the length of a shortest path P from s to t if $DL(P) \leq DL(P')$, where P' is any path from s to t. In the following theorem, we restate the main results of [7].

Theorem 1 [7].

1. A path $P=(s\rightarrow\ldots\rightarrow t)$ has a length $L(P)=M(s,t)+2\cdot DL(P)$.
2. Let P' be a subpath of P from s to x with $DL(P')$.
3. Then $L(P')=M(s,t) + 2\cdot DL(P') - M(x,t)$.
4. If P is a shortest path from s to t, then $DL(P)=\min\{DL(P') \mid P'$ is a path from s to $t\}$.
5. The path generated by the minimum detour algorithm of [7] is a shortest one with the minimized $DL(P)$.

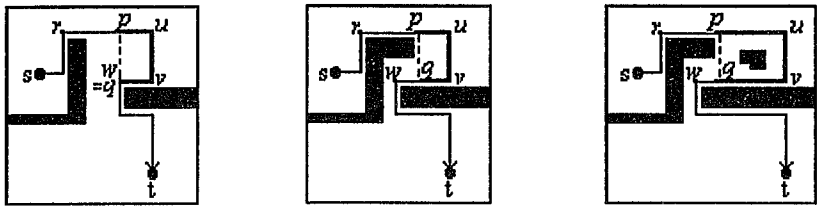
A path P can be represented as a sequence of directed line segments such that no two consecutive line segments have the same direction. A subpath $D=(r\rightarrow u\rightarrow v\rightarrow w)$ in P is called a *detour* (Figure 7.a and 2.b), if directions of the three consecutive line segments $r\rightarrow u$, $u\rightarrow v$, and $v\rightarrow w$ are different. We say that a detour D is *reducible* if

(i) there exists a subpath $R=(p\rightarrow u\rightarrow v\rightarrow q)$ of D where p is on $r\rightarrow u$, q is on $v\rightarrow w$, and $L(p\rightarrow u)=L(v\rightarrow q)>0$,

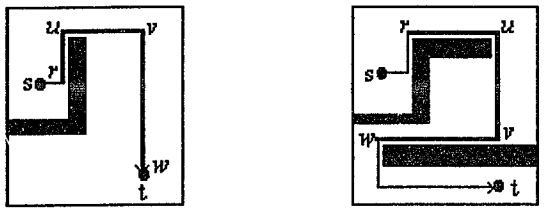
(ii) R is also a detour, and

(iii) R makes the maximum size of rectangle where the edge $p\rightarrow q$ does not intersect any obstacle.

Otherwise, D is a non-reducible. Examples of *reducible detours* are shown in Figure 7.a. Reducible detours should be reduced prior to the generation of the $w\rightarrow\ldots\rightarrow t$ path. The paths $r\rightarrow p\rightarrow q\rightarrow w$ in Figure 7.b are reduced detours. Examples of non-reducible detours are shown in Figure 7.b.

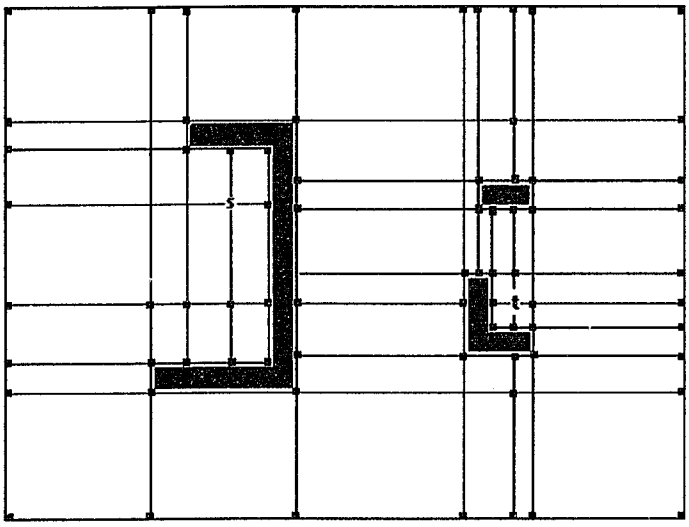


a. Reducible Detours ($r \rightarrow u \rightarrow v \rightarrow w$) and Reduced Detours ($r \rightarrow p \rightarrow q \rightarrow w$)



b. Non-Reducible Detours ($r \rightarrow u \rightarrow v \rightarrow w$)

Figure 7. Detours



■: Base Nodes

Figure 8. G'' and Base Nodes

If a currently extending line segment *l* hits a grid node of *G"* with holding one of the following conditions, the grid node *b* is called a *base node*:

(i) if *b* is on a outer boundary of *G"*,

(ii) if *b* is on a boundary of an obstacle of *B*,

(iii) if *b* is a corner node of *B*, or

(iv) if *b* is on a line passing through *t*.

In addition, the source node is a special case of a base node.

In Figure 8, we show all possible base nodes of an example *G"*.

Our *GMD* algorithm is similar to the *Minimum Detour* (*MD*) algorithm given in [7]. Both of the *GMD* and *MD* algorithms belong to the class of A^* algorithm. For any instance, both the *GMD* and *MD* algorithms find a shortest path from *s* to *t*, if it exists. The differences between them are as follows. The *MD* algorithm is a maze-running algorithm, whereas the *GMD* algorithm is a combination of maze-running and line-search algorithms. The searched space of the *GMD* algorithm is represented by a set of line segments, instead of grid nodes of *G'*. Also, the *GMD* algorithm employs the powerful "*don't change direction*" heuristic along with the use of detour number.

Each line segment $u \rightarrow v$ in *COMPLETE* consists of a 4-tuple (*dir*, *C*, *DL*, *p*), where

(i) *dir* is a direction of $u \rightarrow v$;

(ii) *C* is coordinates of the two end points of $u \rightarrow v$;

(iii) *DL* is a detour length of the subpath $P_s = (s \rightarrow \ldots \rightarrow u \rightarrow v)$, i.e., $DL[u \rightarrow v] = DL(P_s)$; and

(iv) *p* is a pointer that points a predecessor line segment of $u \rightarrow v$.

The line segments are extended as follows. Line segments to be extended are always taken one by one from the queue *OLD*. When a line segment $u \rightarrow v$ is taken from *OLD*, the node *v* is checked as to whether it is a base node or not. If it is a base node, then extensions from *v* to all possible directions are considered. Then, $u \rightarrow v$ is stored in *COMPLETE* and the line segments from *v* to the neighbors ($v \rightarrow w$) are created. If *v* is not a base node, the "*don't change direction*" heuristic is enforced by extending $u \rightarrow v$ to $u \rightarrow w$. Each line segment is extended to one unit or a grid node at a time and controlled by the value of the global detour length *d*. Line extensions from *v* of $u \rightarrow v$ keep proceeding until (i) an extension is directed away from the target node *t*, or (ii) a base node or a visited node is hit. When a line segment is extending one unit away from the target node *t*, the detour length of the line segment is increased by 1. Then, if the detour length of the line segment is greater than *d*, the line segment is added to a queue *NEW* for the next iteration; otherwise, the line

segment continues its extensions. When *OLD* becomes empty, all line segments in *NEW* are moved into *OLD*, d is increased by 1, and *NEW* is reset to empty.

An important operation that reduces the search space is the elimination of the reducible detours defined above. This operation also reduces the search space of the *LGMD* algorithm, which will be explained in section 4. A detour $r \rightarrow u \rightarrow v \rightarrow w$ can be easily detected during the search by tracing back two segments. When such a detour is detected, the procedure *DEL_RD* is called to detect and delete a reducible detour only when $L(v \rightarrow w')$ is less than $L(r \rightarrow u)$, where w' is the first unvisited base node in direction $v \rightarrow w$. The reason *DEL_RD* is called only when $L(v \rightarrow w') < L(r \rightarrow u)$ is that if $L(r \rightarrow u) \leq L(v \rightarrow w')$, a path with a non-reducible detour can be generated before the path $r \rightarrow u \rightarrow v \rightarrow w'$, which may have a reducible detour, is constructed. Let w^* be an intersected point on $v \rightarrow w'$ by a perpendicular line segment from r toward $v \rightarrow w'$ (see Figure 9). There are two cases of $L(r \rightarrow u) \leq L(v \rightarrow w')$:

(i) No obstacle on $r \rightarrow w^*$ (Figure 9.a). The path $r \rightarrow w^*$ has been generated before $r \rightarrow u \rightarrow v \rightarrow w^*$ is constructed, since $DL(r \rightarrow w^*)$ is smaller than $DL(r \rightarrow u \rightarrow v \rightarrow w^*)$.

(ii) Obstacle(s) on $r \rightarrow w^*$ (Figure 9.b). The path $r \rightarrow o \rightarrow p \rightarrow q$ has been generated before $r \rightarrow u \rightarrow v \rightarrow q$ is

constructed, since $DL(r \rightarrow o \rightarrow p \rightarrow q)$ is smaller than $DL(r \rightarrow u \rightarrow v \rightarrow q)$.

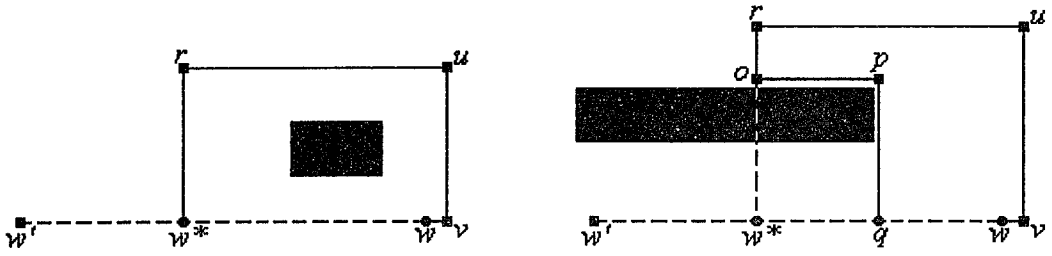
When *DEL_RD* is called, a reducible detour has to be changed to a non-reducible detour. The procedure is to find a line segment $u' \rightarrow v'$ (refer to Figure 10.c) satisfying the following conditions such that:

(i) $u' \rightarrow v'$ is parallel to $u \rightarrow v$,

(ii) $u' \rightarrow v'$ does not intersect any obstacle, and

(iii) the length of $w' \rightarrow v'$ should be minimized.

In the example of Figure 10, nine emanating lines (dotted lines) are generated. If the final emanating line segment, $u' \rightarrow v'$ (refer to Figure 10.c), overlaps $u \rightarrow v$, the detour is not reducible. Otherwise, the reducible detour $r \rightarrow u \rightarrow v \rightarrow w'$ is reduced to $r \rightarrow u' \rightarrow v' \rightarrow w'$ as shown in Figure 10.d. More details of the *GMD* algorithm are given below. Figure 11 shows examples solved by the *GMD* algorithm.



a. No Obstacle on $r \rightarrow w^*$ b. Obstacle(s) on $r \rightarrow w^*$

Figure 9. No Calling the Procedure *DEL_RD* for These Detours $(r \rightarrow u \rightarrow v \rightarrow w)$

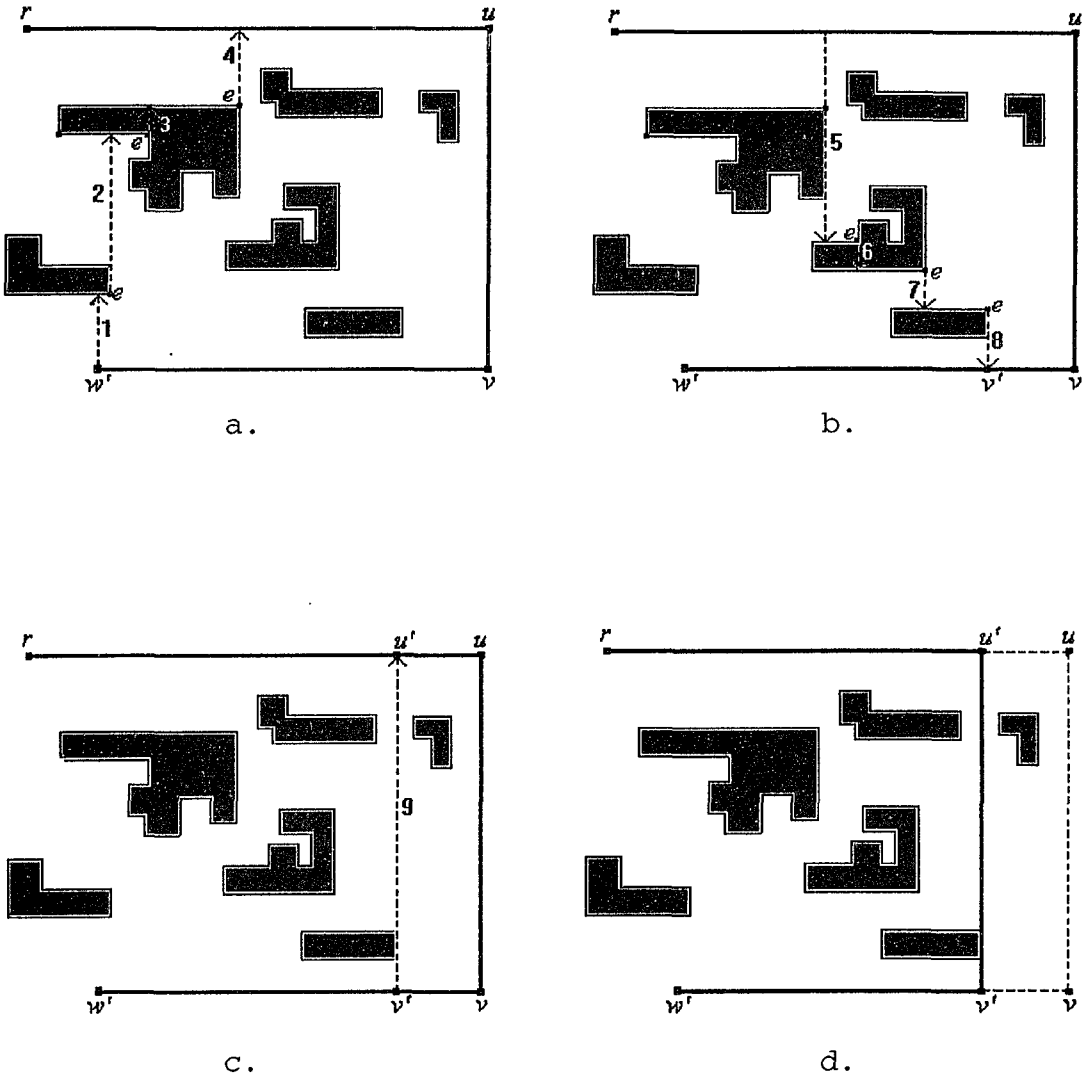


Figure 10. Deleting the Reducible Detour $r \rightarrow u \rightarrow v \rightarrow w'$ to $r \rightarrow u' \rightarrow v' \rightarrow w'$

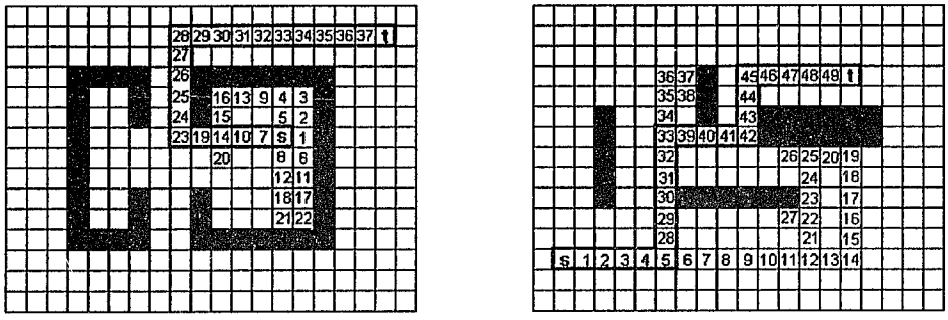


Figure 11. Examples for the *GMD* Algorithm

Guided Minimum Detour Algorithm

// for brevity, "$S \Leftarrow$" and "$S \Rightarrow$ " indicate addition to and deletion from S, respectively //

algorithm *GMD* (s,t);

1 **if** $s = t$ **then** stop;
endif;

2 *NEW* $\Leftarrow$ null; *OLD* $\Leftarrow s \rightarrow s$; *COMPLETE* $\Leftarrow$ null; d:= 0;

3 **while** *OLD* is not empty **do**

4 *OLD* $\Rightarrow u \rightarrow v$;

5 *SEARCH* $(u \rightarrow v)$;
endwhile;

6 **if** *NEW* is empty **then** stop; // no path from s to t exists //
endif;

7 $d := d + 1$;

8 *OLD* := *NEW*; *NEW* := null;

9 **go to** 3;

end *GMD*

```
procedure SEARCH (u→v);
1  if DL[u→v]>d then NEW⇐u→v; //DL[u→v] is a detour length of P=(s→...→u→v)//
2      elseif v is a base node then
3          COMPLETE ⇐ u→v;
4          for each unvisited neighbor node w of v do;
5              create a line segment v→w;
6              if w is t then stop; // a path from s to t is found //
7                  elseif v→w makes a detour r→u→v→w then
8                      if w is an unvisted base node then change v→w to v→w';
9                          else extend v→w to v→w' in direction v→w until a visited
                                   node or an unvisited base node is reached;
                       endif;
10                     if w' is an unvisited base node and L(v→w') < L(r→u) then
11                         v→w := DEL_RD (r→u→v→w');
12                         update DL[v→w] if necessary;
13                         SEARCH (v→w);
14                     else return(); // w' is a visited node //
                       endif;
                   endif;
               endif;
           endfor;
15         elseif a neighbor node w of v in direction u→v is unvisited then
16             if w = t then stop; // a path from s to t is found //
17                 else extend u→v to u→w; // don't change direction //
18                     SEARCH (u→w);
               endif;
           endif;
       endif;
   endif;
19 return();
end SEARCH
```

```
procedure DEL_RD (r→u→v→w'); // deleting reducible detour if exists //
1  flag := 1; v' := w';
2  emanate an orthogonal line from w' toward r→u until a line segment is hit;
   // let the line segment be U //
3  if U hits r→u then // let the hit point be u' //
4     case flag of
5        1: return(u'→v'); 2: return(v'→w');
      endcase;
6     elseif U hits an obstacle then // let the hit obstacle line segment be b→b' //
7        while U hits an obstacle do
8           select one of b and b' close to u→v; // let the selected one be e //
9           emanated an orthogonal line, U, from e toward r→u until a line segment is hit;
         endwhile;
10       if U hits r→u then // let the hit point be u' //
11          emanate an orthogonal line, D, from u' toward v→w' until a line segment is hit;
12          while D hits an obstacle do // let the hit line segment be b→b' //
13             select one of b and b' close to u→v; // let the selected one be e //
14             emanated an orthogonal line, D, from e toward r→u until a line segment is hit;
            endwhile;
15          if D hits v→w' then // let the hit point be v' //
16             emanate an orthogonal line, U, from v' toward r→u until a line segment is hit;
            endif;
         endif;
      endif;
17    flag := 2; goto 3;
   endif;
end DEL_RD
```

CHAPTER 3

ANALYSIS OF THE *GMD* ALGORITHM

For the length of a path from s to t, an obvious lower bound is $M(s,t)$, the Manhattan distance between s and t. By theorem 1, if a path from s to t with length $M(s,t)+2d$, $d \geq 0$, does not exist, where d is a non-negative integer, then the length of shortest path from s to t is at least $M(s,t)+2(d+1)$. Our *GMD* algorithm uses the same principle of the *MD* algorithm, i.e., it exhausts all possible paths of length $M(s,t)+2d$ in G_1" before searching for paths of length $M(s,t)+2(d+1)$ in G_1". By Lemma 1 and Theorem 1, we have the following claim:

Theorem 3. The path $P=(s \rightarrow \ldots \rightarrow t)$ generated by the *GMD* algorithm is an obstacle avoiding shortest path.

As the *MD* algorithm, the *GMD* algorithm belongs to the class of "wave propagation" algorithms (they are also called grid expansion algorithms), even though the searched space of the *GMD* algorithm is represented by a set of line segments. By incorporating the "*don't change direction*" heuristic in our algorithm, the number of wave front line segments in the *GMD* algorithm could be much smaller than the number of wave front nodes in existing maze-running algorithms like the *MD* algorithm, etc. A side effect of

the heuristic, however, is that the extension of a line segment may "overshoot" along its direction so that a "shortcut" may be bypassed. To adjust the bypassing, the *GMD* algorithm deletes reducible detours. Line-search techniques are used in detecting and deleting the reducible detours. During the grid extensions, the *GMD* algorithm generates branches only at base nodes. When a reducible detour is detected, the line segment that forms the "shortcut" overlaps at least one boundary side of an obstacle in B. Therefore, all the line segments generated during the execution of the *GMD* algorithm are on G''. Recall that G'' contains at most $O(e^2)$ edges and G'' is a graph much sparser than G. The performance of the *GMD* algorithm can be expected much better than the *MD* algorithm.

To compare the *GMD* algorithm with the *MD* algorithm, we separate the portion of grid searched by node-by-node grid expansion and the portion searched by detour detection and deletion. The overheads caused by reducible detour detection and deletion will be evaluated shortly. We define the set of searched nodes by the *GMD* (resp. *MD*) algorithm as the set of all grid nodes of G_1'' visited by grid expansion.

Theorem 4. The set of searched nodes by the *GMD* algorithm is a subset of the set of searched nodes by the *MD* algorithm.

Proof. Let S_{GMD} be the set of the searched nodes of the *GMD* algorithm and S_{MD} be the set of the searched nodes of the *MD* algorithm. Let β be a set of all base nodes such that $\beta \subseteq S_{GMD}$. Because of the "*don't change direction*" heuristic used in the *GMD* algorithm, there is at most one choice of extension for every non-base node in S_{GMD} instead of at most three choices for every node in S_{MD} except a start node s. Let g be a node such that $g \in S_{GMD}$ and $g \notin \beta$. Assume there is no obstacle around g. Then, g is also in S_{MD}, since, by the "*don't change direction*," the extended nodes in S_{GMD} are selected from the nodes in S_{MD}. Since g is not a base node, g has only one choice, g', to be extended toward a goal node by the *GMD*. However, g has two choices, g' and g'', toward a goal node by the *MD*. Then $g'' \notin S_{GMD}$ and $g'' \in S_{MD}$. So, $S_{GMD} \subseteq S_{MD}$. □

Now let us analyze the time complexity of the *GMD* algorithm. We difine three basic operations related the *GMD* algorithm:

(i) Given a grid node p of G'' and a direction d, find the first *base node* in *CRITICAL* encountered by a line

emanating from p in direction d. We refer to this operation as *finding the first base node in CRITICAL*.

(ii) Given a grid node p of G'' and a direction d, find the first boundary side of obstacles in B encountered by a line emanating from p in direction d. We refer to this operation as *finding the first element in B*.

(iii) Given a grid node p of G'' and a direction d, find the first segment in *COMPLETE* encountered by a line emanating from p in direction d. We refer to this operation as *finding the first element in COMPLETE*.

First, consider the time for node-by-node extension operations. If we store set of all boundary edges of G' and vertical and horizontal line segments through the target node t, called *CRITICAL*, into the tree-like data structure T_h given in [5], then each operation of *finding the first base node in CRITICAL* can be carried out in $O(\log e)$ time [5], where e is the total number of line segments in *CRITICAL*. If we store all the searched grid nodes of G'' that are on the line segments of *COMPLETE* in a binary search tree T_c, then each operation of *finding the first element in COMPLETE* can be carried out in $O(\log N)$ time, where N is the total number of searched line segments in G''. This operation is used for investigating whether a current line segment hits a line segment in *COMPLETE* or not. Let m be the total number of nodes of G_1'' visited by

grid expansions, i.e., $m=|S_{GMD}|$. Since there are $O(e)$ base nodes among these visited nodes and $O(N)$ line segments composed by these visited nodes, then the total time for grid expansions is $O(m+e\log e+N\log N)$.

The rest of the computations are associated with reducible detour detection and deletion. There are two related basic operations defined above.

(i) *finding the first element in B.*

(ii) *finding the first element in COMPLETE.*

Using the tree-like data structure T_h [5], each operation of finding the first element in B can be carried out in $O(\log e)$ time. T_h is a static data structure, which can be constructed in $O(e\log e)$ time and $O(e)$ space.

Each operation of finding the first element in *COMPLETE* can be carried out in $O(\log N)$ time, where N is the total number of searched line segments in G''. Furthermore, inserting and deleting grid nodes of G'' can be done in $O(\log N)$ time. Therefore, detecting and deleting a reducible detour takes $O(t\log e+\log N)$ time, where t is the number of trial lines (dotted lines in Figure 10.a-c) for a detour. Since the sum of the trial lines for all the detours constructed during the execution of the *GMD* algorithm cannot exceed e, the total time required for processing detours is $O(e\log e+e\log N)$. Taking into account all the time required for grid extensions and manipulating

data structures T_h and T_c, the time complexity of the *GMD* algorithm is $O(m+e\log e+N\log N)$. Since $N=O(e^2)$, the time complexity of the *GMD* algorithm is $O(m+(e+N)\log e)$. The memory space required is $O(e+N)$. On the basis of above analysis, we have the following claim.

Theorem 5. The *GMD* algorithm can be implemented in $O(m+(e+N)\log e)$ time and $O(e+N)$ space, where e is the number of boundary sides of obstacles in B, m is the total number of visited grid nodes of G_1" and N is the total number of searched line segments in G".

Figure 12 shows how the same example in [23] is solved using the four variant maze-running algorithms. The size of their expanded nodes is shown in Figure 13. Figure 14 summarizes some experimental results we have conducted with the randomized obstacles in a 30×40 grid graph. Column 2, *shortest path length*, shows the length of the shortest path for each example. The performance of the *GMD* algorithm is shown in the last two columns. For each of *Lee* algorithm, *Hadlock* algorithm, and *Soukup* algorithm, we give the total number of the expanded nodes, percentage of the searched portion over G, and ratio of the corresponding algorithm over *GMD* with respect to the searched portion, respectively.

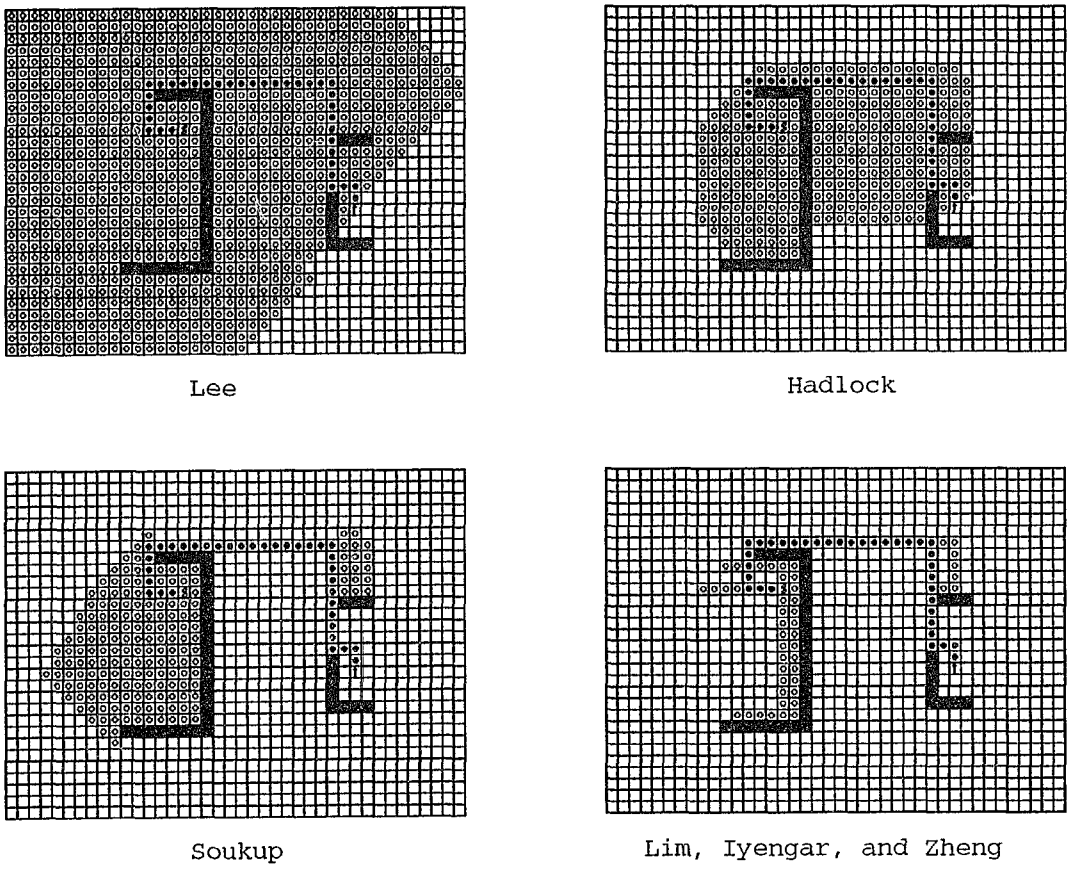


Figure 12. Expanded Nodes of the Four Variants for the Example of Soukup [23]

Example	*Lee*	*Hadlock*	*Soukup*	***Lim, Iyengar, and Zheng (GMD)***
nodes	917	313	215	**79**

Figure 13. Size of Expanded Nodes in Figure 12 for the Four Variants

*Example	Shortest Path Length	*Lee*			*Hadlock*			*Soukup*			*GMD*	
		# of Searched Nodes	% of Searched Portion	*Lee* / *GMD*	# of Searched Nodes	% of Searched Portion	*Hadlock* / *GMD*	# of Searched Nodes	% of Searched Portion	*Soukup* / *GMD*	# of Searched Nodes	% of Searched Portion
1	35	917	79%	11.3	313	27%	3.9	215	19%	2.7	79	7%
2	36	1060	95%	20.0	566	51%	10.7	244	21%	4.2	53	5%
3	48	1079	96%	6.4	700	62%	4.1	323	29%	1.9	169	15%
4	53	1093	97%	7.5	673	60%	4.6	573	51%	3.9	152	13%
5	54	1067	94%	5.3	859	74%	4.2	440	39%	2.2	202	18%
6	59	1101	96%	5.7	774	68%	4.0	387	34%	2.0	193	17%
7	67	942	83%	4.2	679	60%	3.0	511	45%	2.3	228	20%
8	71	921	84%	4.4	680	62%	3.3	609	56%	2.9	207	19%
9	72	1024	93%	4.7	531	48%	2.4	404	37%	1.9	225	20%
10	74	1070	96%	5.6	813	73%	4.3	812	73%	4.3	185	17%
11	78	1126	95%	7.3	823	70%	5.4	836	71%	5.5	150	13%
12	150	1087	97%	4.0	966	86%	3.6	881	78%	3.3	265	24%
Average	66	1041	92%	7.2	698	62%	4.5	520	46%	3.1	176	16%

*using 30x40 grid graph with randomized obstacles

Figure 14. Comparisons of the Experimental Results

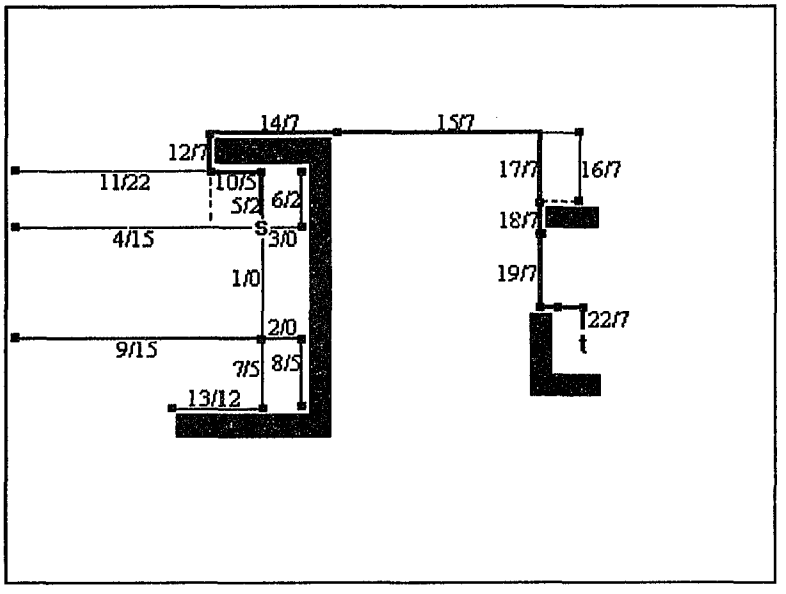
CHAPTER 4

A MODIFIED ALGORITHM: LINE-BY-LINE GUIDED MINIMUM DETOUR ALGORITHM (*LGMD*)

Let us now consider a modification of the *GMD* algorithm. In the *GMD* algorithm, the line segments in *OLD* are extended node by node and moved immediately into *NEW* after the line segments are extended away from the target node t. Now, without losing the general features of the *GMD* algorithm, we contemplate *line-by-line* extensions rather than node-by-node extensions to generate line segments. Each line segment in *COMPLETE* must be from a base node to a base node except the line segment constructed by deleting reducible detour. In other words, a line segment is extended until a base node is hit. A 4-tuple (dir, C, DL, p) information (refer to the definition in chapter 2) is assigned to each extended line segment $u \rightarrow v$. The line segment that has the lowest detour length will be chosen for the next extensions. To implement this modification, we use a *priority queue*, called *OPEN*, to select the line segment that has the lowest detour length instead of the queues *OLD* and *NEW* in the *GMD* algorithm. By the queue *OPEN*, the global variable d, detour length, in the *GMD* algorithm is not needed. Such a modified algorithm is called the *Line-by-Line Guided Minimum Detour* (*LGMD*)

algorithm. The *LGMD* algorithm not only compromises the existing *GMD* algorithm's drawback--the running time--but also shares the solution optimality of the *GMD* algorithm.

Following are the detailed procedures of the modified *LGMD* algorithm including the above operations. For the same example in Figure 12, the generated whole line segments with sequence numbers and detour lengths by the *LGMD* algorithm are shown in Figure 15.



--- :Trial Line for Deleting Reducible Detour, ■ : Base Node
n_1/n_2 = Order of Extensions/Detour Length

Figure 15. Extended Line Segments for the *LGMD* Algorithm

Line-by-Line Guided Minimum Detour (LGMD) Algorithm

// for brevity, "$S \Leftarrow$ " and "$S \Rightarrow$ " indicate addition to or deletion from S, respectively //
algorithm *LGMD* (s,t);

1 **if** $s = t$ **then** stop;
endif;

2 $OPEN \Leftarrow s \rightarrow s$; $COMPLETE \Leftarrow$ null;

3 **while** *OPEN* is not empty **do**

4 $OPEN \Rightarrow u \rightarrow v$; $COMPLETE \Leftarrow u \rightarrow v$;

5 *SEARCH* ($u \rightarrow v$);
endwhile;

6 **if** *OPEN* is empty **then stop**; // no path from s to t exists //
end *LGMD*

procedure *SEARCH_L* ($u \rightarrow v$);
// let b be the set of nearest unvisited base nodes from v in all possible directions //
1 **for** each base node w' in b **do**;

2 **if** there is no intersections on $v \rightarrow w'$ **then** create a line segment $v \rightarrow w'$;

3 **if** w' is t **then** stop; // a path from s to t is found //

4 **elseif** $v \rightarrow w'$ makes a detour $r \rightarrow u \rightarrow v \rightarrow w'$ **then**

5 **if** $L(v \rightarrow w') < L(r \rightarrow u)$ **then**

6 $v \rightarrow w' := DEL\_RD\ (r \rightarrow u \rightarrow v \rightarrow w')$;

7 update $DL[v \rightarrow w]$ if necessary;

8 $OPEN \Leftarrow v \rightarrow w'$;
endif;

9 **else** $OPEN \Leftarrow v \rightarrow w'$;
endif;
endif;
endif;
endfor;

10 **return**();
end *SEARCH_L*

By an analysis similar to that of the *GMD* algorithm, we conclude the performance of the *LGMD* algorithm by the following theorem.

Theorem 6. The *LGMD* algorithm can be implemented in $O((e+N)\log e)$ time and $O(e+N)$ space, where e is the number of boundary sides of obstacles in B and N is the total number of searched line segments in G''.

CHAPTER 5

A COMBINED LENGTH AND BENDS SHORTEST PATH

The objective of this chapter is to develop an efficient combined length and bends shortest path problem using the *LGMD* algorithm shown in chapter 4. The number of bends on paths gains more attention recently [2][25]. The current shortest path algorithms find a shortest path but leaves the number of bends in the solution path uncertain. Yang et al. [26] provide a unified approach by constructing a *path-preserving graph* guaranteed to preserve all these kinds of paths and give an $O(k+e\log e)$ algorithm to find them, where e is the total number of obstacle edges, and k is the number of intersections between *tracks* from *extreme point* and other tracks. k is bounded $O(ne)$ where n is the number of obstacle. We will consider, specifically, the problems of finding a minimum-bend shortest path, a shortest minimum-bend path without constructing any track graph. In the dynamic environment like mobile obstacles, the track graph (*path-preserving*) have to be reconstructed whenever any obstacle is moved. However, the data structure for *LGMD* without track graph needs only a few operations of insertion or deletion for line segments of a moved or changed obstacle. The set of problems to be

considered in this chapter for shortest paths are as follows (refer to Figure 16):

(i) *LGMD_MB*: a path with a minimum number of bends

(ii) *LGMD_MBS*: a path with minimum-bend path whose length is shortest

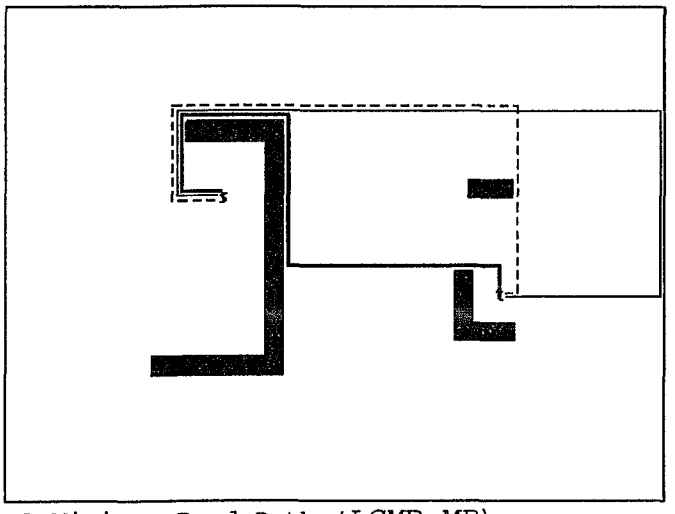
(iii) *LGMD_SMB*: a shortest path with minimum-bend path

The procedures for the *LGMD_MB* and *LGMD_SMB* are similar to the *LGMD* algorithm in chapter 4. Let us discuss the *LGMD_MB* algorithm. Each line segment in *COMPLETE* must be from a base node to a base node. For each line segment $u \rightarrow v$ in *COMPLETE*, a 4-tuple (*dir*, *C*, *MB*, *p*) information (refer to the definition in chapter 2 for *dir*, *C*, and *p*) is assigned to each extended line segment $u \rightarrow v$, where *MB* is a number bends of a path $P=(s \rightarrow \ldots \rightarrow u \rightarrow v)$, i.e., $MB[u \rightarrow v]=MB(P)$.

The line segment that has the lowest number of bends will be chosen for the next extensions. We use a *priority queue*, called *OPEN*, to select the line segment that has the lowest *MB* as in the *LGMD* algorithm. Such a modified algorithm is called the *LGMD_MB* algorithm. The difference from the *LGMD* algorithm is that we substitute *DL* to *MB* as a lower bound.

Following are the detailed procedures of the *LGMD_MB* algorithm. For the same example in Figure 12, the generated whole line segments with generated sequence

numbers and *MB* by the *LGMD_MB* algorithm are shown in Figure 17.



— A Minimum-Bend Path (*LGMD_MB*)
▬ A Shortest Minimum-Bend Path (*LGMD_SMB*)
- - A Minimum-Bend Shortest Path (*LGMD_MBS*)

Figure 16. Example of Different Shortest Paths

LGMD_MB Algorithm

// for brevity, "$S \Leftarrow$ " and "$S \Rightarrow$ " indicate addition to or deletion from S, respectively //
algorithm *LGMD_MB* (*s,t*);

1 **if** $s = t$ **then** stop;
endif;

2 *OPEN* $\Leftarrow s \rightarrow s$; *COMPLETE* $\Leftarrow$ null;

3 **while** *OPEN* is not empty **do**

4 *OPEN* $\Rightarrow u \rightarrow v$; *COMPLETE* $\Leftarrow u \rightarrow v$;

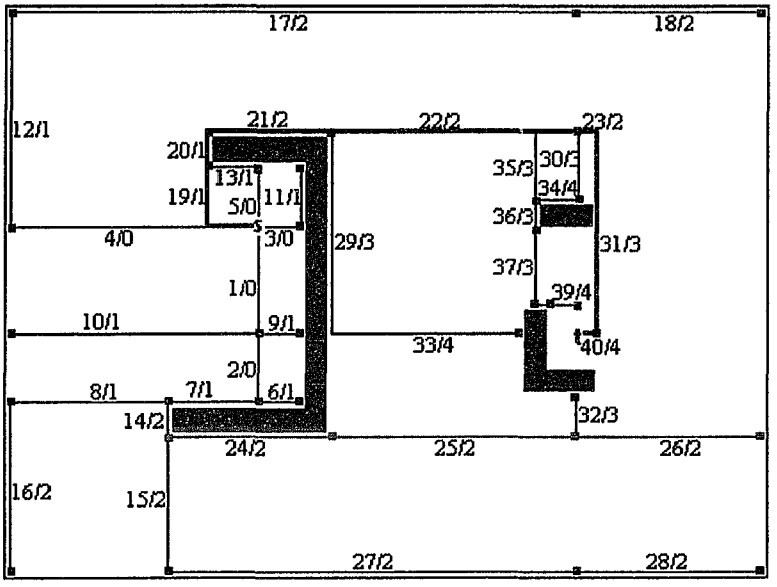
5 *SEARCH* ($u \rightarrow v$);
endwhile;

6 **if** *OPEN* is empty **then stop**; // no path from s to t exists //
end *LGMD_MB*

```
procedure SEARCH_MB (u→v);
// let b be the set of nearest unvisited base nodes from v in all possible directions //
1  for each base node w' in b do;
2     if there is no intersections on v→w' then create a line segment v→w';
3        if w' is t then stop; // a path from s to t is found //
4           elseif v→w' makes a detour r→u→v→w' then
5              if L(v→w') < L(r→u) then
6                 v→w' := DEL_RD (r→u→v→w');
7                 update MB[v→w] if necessary;
8                 OPEN ⇐ v→w';
                endif;
9           else OPEN ⇐ v→w';
            endif;
        endif;
     endif;
  endfor;
10 return();
end SEARCH_MB
```

By an analysis similar to that of the *LGMD* algorithm, we conclude the performance of the *LGMD_MB* algorithm by the following theorem.

Theorem 7. The *LGMD_MB* algorithm can be implemented in $O((e+N)\log e)$ time and $O(e+N)$ space, where e is the number of boundary sides of obstacles in B and N is the total number of searched line segments in G''.



■ : Base Node, n_1/n_2=Order of Extensions/*MB*

Figure 17. Extended Line Segments for the *LGMD_MB* Algorithm

The procedures for the *LGMD_MBS* algorithm is same to the *LGMD_MB* algorithm except the lower bound. For each line segment $u \rightarrow v$ in *COMPLETE*, a 5-tuple (*dir*, *C*, *DL*, *MB*, *p*) information is assigned to each extended line segment $u \rightarrow v$. Among the line segments that have the lowest *MB*, a line segment with the lowest *DL* will be chosen for the next extensions.

Similarly, the procedures for the *LGMD_SMB* algorithm can find a shortest path with minimum number of bends using a 5-tuple (*dir*, *C*, *DL*, *MB*, *p*) information for each line segment $u \rightarrow v$ in *COMPLETE*. Among the line segments that

have the lowest *DL*, a line segment with the lowest *MB* will be chosen for the next extensions.

By an analysis similar to that of the *LGMD_MB* algorithm, the performance of the *LGMD_MBS* algorithm and the *LGMD_SMB* algorithm are concluded by the following theorem.

Theorem 8. The *LGMD_MBS* (or *LGMD_SMB*) algorithm can be implemented in $O((e+N)\log e)$ time and $O(e+N)$ space, where e is the number of boundary sides of obstacles in B and N is the total number of searched line segments in G''.

CHAPTER 6

SUMMARY AND CONCLUSIONS

We introduced a heuristic approach to find rectilinear (L_1) shortest path with presence of obstacles. The *GMD* algorithm combines the best features of maze-running algorithms and line-search algorithms. The *LGMD* algorithm is a modification of the *GMD* algorithm that improves on its efficiency. A comparison of the new algorithms with the existing algorithms is presented in Figure 18.

	Lee	*Hadlock*	*Wu et al.*	*GMD*	*LGMD*
Time	$O(n^2)$	$O(n^2)$	$O((e+k)\log t)$	$O(m+(e+N)\log e)$	$O((e+N)\log e)$
Space	$O(n^2)$	$O(n^2)$	$O(e+k)$	$O(e+N)$	$O(e+N)$
Search Method	Breadth-First	A*	Dijkstra's Search	Grid-by-Grid Guided A*	Line-by-Line Guided A*
Solution Optimality	Optimal	Optimal	Suboptimal	Optimal	Optimal
Connection Graph	Grid Graph	Grid Graph	Track Graph	Not Needed	Not Needed

Figure 18. Bounds on the Algorithms Discussed in the Previous Sections

Let us compare the *LGMD* algorithm with the algorithm given by Wu et al. [24]. Before the search for a shortest path from s to t starts, the algorithm in [24] constructs a grid-like track graph G_T. The space for storing G_T is

$O(e+k)$, and the time for constructing G_T and finding a shortest path from s to t is $O((e+k)\log t)$, where e is the total number of boundary sides of obstacles, k is the number of nodes in G_T, and t is the total number of extreme edges in the obstacles (for the definition of extreme edges, refer to [24]). Our *LGMD* algorithm takes $O(e+N)$ space and $O((e+N)\log e)$ time. In the worst case, $t=O(e)$, $k=O(e^2)$, and the space and time complexities of the algorithm in [24] are $O(e^2)$ and $O(e^2 \log e)$. The performance of our *LGMD* algorithm depends on N, the total number of searched edges in G''. Even though in the worst case G'' contains $O(e^2)$ edges, since our *LGMD* algorithm does not have a preprocessing phase for generating G'', the total number N of searched edges tends to be much smaller than $O(e^2)$. The use of detour length, "*don't change direction*" heuristic for guided A^*, and reducible detour deletion operations for finding "shortcuts" is another factor resulting a small N. Therefore, our *LGMD* algorithm can be expected to outperform the algorithm given in [24].

Since the detour length as a lower bound in our algorithms can be substituted for the number of bends in the rectilinear link metric [2][11][25] or the channel wiring density [3], our algorithms can be easily extended to these problems. We consider the problem of finding a

shortest path in terms of the number of bends and combined length and bends in chapter 5.

Our heuristic approach is designed for one-time query. If, however, the repetitive mode is needed in some applications, the heuristic search method in both the *GMD* and the *LGMD* algorithm can be performed on a connection graph G'' for the repetitive-mode queries [26].

BIBLIOGRAPHY

[1]S. B. Akers, "*A Modification of Lee's Path Connection Algorithm,*" IEEE Transactions on Electronic Computers, EC-16(2), pp. 97-98, 1967.

[2]M. T. De Berg, M. J. Van Kreveld, B. J. Nilsson, and M. H. Overmars, "*Finding Shortest Paths in the Presence of Orthogonal Obstacles Using a Combined L_1 and Link Metric,*" in Proceedings of the Second Scandinavian Workshop on Algorithm Theory, pp. 213-24, 1990.

[3]A. D. Brown and M. Zwolinski, "*Lee Router Modified for Global Routing,*" Computer-Aided Design, Vol. 22, No. 5, pp. 296-300, June, 1990.

[4]K. L. Clarkson, S. Kapoor, and P. M. Vaidya, "*Rectilinear Shortest Paths through Polygonal Obstacles in $O(n(\log n)^2)$ Time,*" in Proceedings of the Third Annual Conference on Computational Geometry, pp. 251-57, ACM, 1987.

[5]H. Edelsbrunner and M. H. Overmars, "Some Methods of Computational Geometry Applied to Computer Graphics," Computer Vision, Graphics, and Image Processing, Vol. 28, pp. 92-108, 1984.

[6]J. M. Geyer, "*Connection Routing Algorithms for Printed Circuit Boards,*" IEEE Transactions on Circuit Theory, CT-18(1), pp. 95-100, 1971.

[7]F. O. Hadlock, "*The Shortest Path Algorithm for Grid Graphs,*" Networks, 7, pp. 323-34, 1977.

[8]P. Hart, N. Nilsson and B. Raphael, "*A Formal Basis for the Heuristic Determination of Minimum Cost Paths,*" IEEE Transactions on Systems, Science and Cybernetics, SCC-4(2), pp. 100-107, 1968.

[9]D. W. Hightower, "*A Solution to Line Routing Problems on the Continuous Plane,*" in Proceedings of the Sixth Design Automation Workshop, pp. 1-24, IEEE, 1969.

[10] J. H. Hoel, "*Some Variation of Lee's Algorithm,*" IEEE Transactions on Computers, C-25(1), pp. 19-24, 1976.

[11] Y. Ke, "*An Efficient Algorithm for Link-Distance Problems*," in Proceedings of 5th ACM Symp., Lect. Notes in Computer Science, 447, Springer-Verlag, pp. 213-224, 1990.

[12] C. Y. Lee, "*An Algorithm for Path Connections and Its Applications*," IRE Transactions on Electronic Computers, EC-10(3), pp. 346-65, 1961.

[13] T. Lengauer, "Combinatorial Algorithms for Integrated Circuit Layout," Wiley, Reading, England, 1990.

[14] J. S. Lim, S. S. Iyengar, and S.-Q. Zheng, "*Rectilinear Shortest Path Problem with Rectilinear Obstacles*," in Proceedings of the Sixth International Conference on VLSI Design, pp. 90-93, Jan. 1993.

[15] K. Mikami, K. Tabuchi, "*A Computer Program for Optimal Routing of Printed Circuit Connectors*," IFIPS Proceedings, H-47, pp. 1475-78, 1968.

[16] J. S. B. Mitchell, "*An Optimal Algorithm for Shortest Rectilinear Paths Among Obstacles in the Plane*," in Abstracts of the First Canadian Conference on Computational Geometry, p. 22, 1989.

[17] E. F. Moore, "The Shortest Path Through a Maze," Annals of the Harvard Computation Laboratory, Vol. 30, Pt. II, pp. 285-92, 1959.

[18] T. Ohtsuki, "*Maze-running and Line-search Algorithms*," in T. Ohtsuki, editor, Advances in CAD for VLSI, Vol. 4: Layout Design and Verification, pp. 99-131, North-Holland, New York, 1986.

[19] I. Pohl, "*Heuristic Search Viewed as Path Finding in a Graph*," Artificial Intelligence, Vol. 1, pp. 193-204, 1970.

[20] ___, "*Bi-Directional Search*," Machine Intelligence, Vol. 6, pp. 127-40, 1971.

[21] P. J. Rezend, D. T. Lee, and Y.-F. Wu, "Rectilinear Shortest Paths with Rectangular Barriers," in Proceedings of the Second Annual Conference on Computational Geometry, pp. 204-13, ACM, 1985.

[22] F. Rubin, "*The Lee Path Connection Algorithm*," IEEE Transactions on Computers, C-23(9), pp. 907-14, 1974.

[23] J. Soukup, "*Fast Maze Router*," in Proceedings of the 15th Design Automation Conference, pp. 100-102, 1978.

[24] Y.-F. Wu, P. Widmayer, M. D. F. Schlag, C. K. Wong, "*Rectilinear Shortest Paths and Minimum Spanning Trees in the Presence of Rectilinear Obstacles*," IEEE Transactions on Computers, C-36(3), pp. 321-31, 1987.

[25] C. D. Yang, D. T. Lee, and C. K. Wong, "*On Bends and Lengths of Rectilinear Paths: A Graph-Theoretic Approach*," in Proceedings of Algorithms and Data Structures, 2nd Workshop WADS '91, Lect. Notes in Computer Science, 519, Springer-Verlag, pp. 320-330, 1991.

[26] S.-Q. Zheng, J. S. Lim, and S. S. Iyengar, "*Efficient Maze-Running and Line-Search Algorithms for VLSI Layout*," in Proceedings of the IEEE Southeastcon '93, Session M4B, IEEE, 1993.

VITA

I was born in Korea in Seoul, Korea, in 1959. In my early school years I attended Kyo-Dong Primary School. After my graduation I entered the Yun-Hee Middle School. Upon graduation from Hong-Ik High School I entered Inha University. I majored in Computer Science. After graduation I planned to study abroad in America. I was conferred a M.S. Degree from the University of Alabama at Birmingham in June 1989. My interested area was artificial intelligence. During the period I published two papers:

- J. S. Lim, "Parallel Hill-Climbing Search," in Proceedings of the 27th Annual Conference of the Southeast Region, ACM, pp. 646-649, April, 1989.
- P. Dey, D. R. Whitlock, and J. S. Lim, "Parallel Best-First Search Strategies," in Proceedings of the 4th ISMM/IASTED International Conference on Parallel and Distributed Computing and Systems, Session 26-Parallel Algorithm III, October, 1991.

Soon Afterwards I was transferred to LSU for Ph. D. program, I found myself more in the study of computer science. The following published papers shows my interested subject:

- S.-Q. Zheng, J. S. Lim, and S. S. Iyengar, "*Efficient Maze-Running and Line-Search Algorithms for VLSI Layout*," in Proceedings of the IEEE Southeastcon '93, Session M4B, IEEE, 1993.
- J. S. Lim, S. S. Iyengar, and S.-Q. Zheng, "*Rectilinear Shortest Path Problem with Rectilinear Obstacles*," in Proceedings of the Sixth International Conference on VLSI Design, pp. 90-93, Jan. 1993.

After I obtain the degree, I want to return to Korea and will teach a younger generation of Korea.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Joon Shik Lim

Major Field: Computer Science

Title of Dissertation: A Heuristic Approach for Shortest Path Problem with Rectilinear Obstacles

Approved:

Major Professor and Chairman

Dean of the Graduate School

EXAMINING COMMITTEE:

Date of Examination:

March 18, 1994